```
#Project Ridge Regression
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import cross_val_predict
import statsmodels.api as sm
import seaborn as sns
from sklearn.preprocessing import OneHotEncoder
from sklearn.linear_model import Ridge
```

```
data = pd.read_csv("dataset.csv")
```

```
#checking the dataset
```

```
data.describe()
```

	Unnamed: 0	popularity	duration_ms	danceability	energy	
count	114000.000000	114000.000000	1.140000e+05	114000.000000	114000.000000	11400(
mean	56999.500000	33.238535	2.280292e+05	0.566800	0.641383	!
std	32909.109681	22.305078	1.072977e+05	0.173542	0.251529	(
min	0.000000	0.000000	0.000000e+00	0.000000	0.000000	(
25%	28499.750000	17.000000	1.740660e+05	0.456000	0.472000	;
50%	56999.500000	35.000000	2.129060e+05	0.580000	0.685000	!
75%	85499.250000	50.000000	2.615060e+05	0.695000	0.854000	{
max	113999.000000	100.000000	5.237295e+06	0.985000	1.000000	1

```
data.head()
```

	Unnamed: 0	track_id	artists	album_name	track_name	]
0	0	5SuOikwiRyPMVoIQDJUgSV	Gen Hoshino	Comedy	Comedy	
1	1	4qPNDBW1i3p13qLCt0Ki3A	Ben Woodward	Ghost (Acoustic)	Ghost - Acoustic	
2	2	1iJBSr7s7jYXzM8EGcbK5b	Ingrid Michaelson;ZAYN	To Begin Again	To Begin Again	
3	3	6lfxq3CG4xtTiEg7opyCyx	Kina Grannis	Crazy Rich Asians (Original Motion Picture Sou...	Can't Help Falling In Love	
4	4	5vjLSffimiIP26QG5WcN2K	Chord Overstreet	Hold On	Hold On	

5 rows × 21 columns

```
#spliting a data into a test and training set
```

```
train, test = train_test_split(data, test_size=.2, random_state = 1)
```

```
#defining a Ridge model for numerical variables model for alpha =1
```

```
predictors = ['duration_ms', 'danceability' , 'energy', 'key',
              'loudness', 'mode','speechiness', 'acousticness', 'instrumentalness', 'liveness',
              'valence', 'tempo', 'time_signature']
target = 'popularity'
```

```
X = train[predictors].copy()
y = train[[target]].copy()
```

```
x_mean = X.mean()
x_std = X.std()
```

```
X = (X - x_mean) / x_std
```

```
X.describe()
```

	duration_ms	danceability	energy	key	loudness	
count	9.120000e+04	9.120000e+04	9.120000e+04	9.120000e+04	9.120000e+04	9.1
mean	1.781713e-17	9.963599e-15	-3.481450e-14	-3.579812e-16	4.167578e-15	-5.
std	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.0
min	-2.042138e+00	-3.261479e+00	-2.545923e+00	-1.490821e+00	-8.173745e+00	-1.3
25%	-5.024361e-01	-6.374353e-01	-6.718861e-01	-9.293504e-01	-3.472165e-01	-1.3
50%	-1.408392e-01	7.612053e-02	1.738129e-01	-8.714487e-02	2.503902e-01	7.
75%	3.096426e-01	7.378860e-01	8.448136e-01	7.550606e-01	6.467478e-01	7.
max	4.660786e+01	2.406686e+00	1.424495e+00	1.597266e+00	2.535042e+00	7.

```
X["intercept"] = 1
X = X[["intercept"] + predictors]
```

```
X.T
```

	104483	17411	73414	95288	77403	1216	6
intercept	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.00
duration_ms	0.066370	-0.382093	0.900388	0.450949	-0.123068	-0.417338	0.28
danceability	-0.867615	0.645814	1.134945	1.129191	-0.125286	0.116402	0.61
energy	-1.124514	1.233915	-1.454058	0.666145	0.169843	0.177783	0.89
key	1.316531	-1.210086	-1.490821	-1.490821	0.474325	1.035796	0.47
loudness	-0.053266	1.087277	-0.699006	0.655463	0.075882	0.624563	0.97
mode	0.756612	0.756612	-1.321667	0.756612	0.756612	-1.321667	0.75
speechiness	-0.364595	-0.237009	-0.429340	-0.368403	-0.371260	0.357120	0.82
acousticness	0.482657	-0.948229	-0.214774	-0.274897	1.005730	-0.935413	-0.87
instrumentalness	-0.504215	1.710293	1.855560	-0.504215	-0.504215	-0.481456	-0.50
liveness	-0.538002	1.582232	0.369920	-0.699119	2.626605	-0.895922	-0.61
valence	-1.380735	-1.153252	-0.143070	0.828556	1.256534	0.909525	-0.17
tempo	-1.381716	0.199269	-0.235327	-1.182783	1.268994	2.064323	0.19
time_signature	-2.101340	0.221990	0.221990	0.221990	0.221990	0.221990	0.22

14 rows × 91200 columns

```
alpha = 1
I = np.identity(X.shape[1])
```

```
I
```

```
array([[1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1.]])
```

```
I [0][0] = 0
```

```
I
```

```
array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1.]])
```

```
penalty = alpha * I
```

```
print(penalty)
```

```
array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1.]])
```

```
B = np.linalg.inv(X.T @ X + penalty) @ X.T @ y
```

```
B
```

	popularity
0	33.235570
1	-0.235500
2	1.545515
3	-0.766560
4	-0.089496
5	0.628311
6	-0.392757
7	-1.306663
8	-0.261086
9	-2.525821
10	0.327430
11	-2.489879
12	0.423009
13	0.464909

```
#Adding raw labels into a df
B.index = ['intersept', 'duration_ms', 'danceability' , 'energy', 'key',
              'loudness', 'mode','speechiness', 'acousticness', 'instrumentalness', 'liveness',
              'valence', 'tempo', 'time_signature']
print(B)
```

```
                   popularity
intersept           33.235570
duration_ms         -0.235500
danceability         1.545515
energy              -0.766560
key                 -0.089496
loudness             0.628311
mode                -0.392757
```

```
speechiness        -1.306663
acousticness       -0.261086
instrumentalness   -2.525821
liveness            0.327430
valence            -2.489879
tempo               0.423009
time_signature      0.464909
```

```
test_X = test[predictors]
test_X = (test_X - x_mean)/x_std
test_X["intercept"] = 1
test_X = test_X[["intercept"] + predictors]
```

```
test_X
```

	intercept	duration_ms	danceability	energy	key	loudness	
21719	1	0.110035	1.629831	0.074552	1.597266	0.677450	-1.3
25741	1	-0.176540	0.421389	0.463653	1.035796	0.189382	-1.3
15850	1	-0.603221	0.835712	-1.168188	-1.490821	-0.415356	-1.3
31992	1	0.071208	0.686096	1.075098	1.316531	0.961695	0.7
79027	1	-0.287700	-1.753805	0.142050	0.474325	0.519184	0.7
...	...	...	...	...	...	...	
58712	1	0.496233	0.087629	0.646293	-1.490821	0.665764	0.7
67442	1	0.204075	-0.620172	-0.417779	-0.087145	0.496405	0.7
36107	1	-0.501106	1.169472	0.928192	1.316531	0.842649	-1.3
40989	1	0.415257	-0.562627	0.360423	-0.367880	0.709143	0.7
30180	1	-0.422800	0.030085	0.431890	-1.210086	0.638627	-1.3

22800 rows × 14 columns

```
B
```

	popularity
intersept	33.235570
duration_ms	-0.235500
danceability	1.545515
energy	-0.766560
key	-0.089496
loudness	0.628311
mode	-0.392757
speechiness	-1.306663
acousticness	-0.261086
instrumentalness	-2.525821
liveness	0.327430
valence	-2.489879
tempo	0.423009
time_signature	0.464909

```
predictions = np.dot( test_X , B)
print(predictions)
```

```
[[35.00710664]
 [31.47223575]
 [37.42379568]
 ...
 [34.03764031]
 [34.46233612]
 [36.51388306]]
```

```
print(predictions.mean())
print(predictions.std())
```

```
print(predictions.min())
print(predictions.max())
```

```
33.23697795881275
3.4022706987961735
12.871222385472889
43.731512367086026
```

```
# Ridge Regression for alpha = 100
```

```
alpha2= 100
```

```
penalty2= alpha2 * I
```

```
B2 = np.linalg.inv(X.T @ X + penalty2) @ X.T @ y
print(B2)
predictions2 = np.dot( test_X, B2)
print(predictions2)
```

```
    popularity
0    33.235570
1    -0.234971
2     1.541239
3    -0.765788
4    -0.089449
5     0.628707
6    -0.392373
7    -1.304416
8    -0.260812
9    -2.521707
10    0.326096
11   -2.483956
12    0.421813
13    0.464700
[[35.00407318]
 [31.47828424]
 [37.41629648]
 ...
 [34.03698484]
 [34.46212591]
 [36.50872436]]
```

```
print(predictions2.mean())
print(predictions2.std())
print(predictions2.min())
print(predictions2.max())
```

```
33.236968219062156
3.3964004811906485
12.90016031886273
43.706083987053844
```

```
B2.index = ['intersept', 'duration_ms', 'danceability' , 'energy', 'key',
              'loudness', 'mode','speechiness', 'acousticness', 'instrumentalness', 'liveness',
              'valence', 'tempo', 'time_signature']
print(B2)
```

```
                  popularity
intersept          33.235570
duration_ms        -0.234971
danceability        1.541239
energy             -0.765788
key                -0.089449
loudness            0.628707
mode               -0.392373
speechiness        -1.304416
acousticness       -0.260812
instrumentalness   -2.521707
liveness            0.326096
valence            -2.483956
tempo               0.421813
time_signature      0.464700
```

```
# Ridge Regression for alpha = 10 000
```

```
alpha3= 10000
```

```
penalty3= alpha3 * I
```

```
B3 = np.linalg.inv(X.T @ X + penalty3) @ X.T @ y
print(B3)
```

```
    popularity
0    33.235570
1    -0.193455
2     1.211093
3    -0.662612
4    -0.084016
5     0.624330
6    -0.355046
7    -1.120423
8    -0.228239
9    -2.181532
10    0.220558
11   -2.009433
12    0.328394
13    0.439915
```

```
B3.index = ['intersept', 'duration_ms', 'danceability' , 'energy', 'key',
              'loudness', 'mode','speechiness', 'acousticness', 'instrumentalness', 'liveness',
              'valence', 'tempo', 'time_signature']
print(B3)
predictions3 = np.dot( test_X , B3)
predictions3
```

```
                  popularity
intersept          33.235570
duration_ms        -0.193455
danceability        1.211093
energy             -0.662612
key                -0.084016
loudness            0.624330
mode               -0.355046
speechiness        -1.120423
acousticness       -0.228239
instrumentalness   -2.181532
liveness            0.220558
valence            -2.009433
tempo               0.328394
time_signature      0.439915
array([[34.75142259],
       [31.96926723],
       [36.7761786 ],
       ...,
       [33.99581028],
       [34.42451761],
       [36.06814109]])
```

```
print(predictions3.mean())
print(predictions3.std())
print(predictions3.min())
print(predictions3.max())
```

```
33.23617875436611
2.9103166117590593
15.396092371329223
41.65279669207298
```

```
# Ridge Regression for alpha = 1 000 000


alpha4= 1000000


penalty4= alpha4 * I


B4 = np.linalg.inv(X.T @ X + penalty4) @ X.T @ y
print(B4)
predictions4 = np.dot( test_X, B4)
print(predictions4)
```

```
    popularity
0    33.235570
1    -0.015005
2     0.069715
3    -0.009426
4    -0.006755
5     0.086698
6    -0.029476
7    -0.082581
8    -0.036006
9    -0.177366
10   -0.006666
11   -0.081366
```

```
12    0.020702
13    0.055113
[[33.39780725]
 [33.28465149]
 [33.44893759]
 ...
 [33.41500757]
 [33.3909247 ]
 [33.49612551]]
```

```
print(predictions4.mean())
print(predictions4.std())
print(predictions4.min())
print(predictions4.max())
```

```
33.23591696146896
0.26433927008203156
31.59694998634919
33.719962225612164
```

```
#errors for every model built
```

```
def ridge_fit(train, predictors, target, alpha):
    X = train[predictors].copy()
    y = train[[target]].copy()

    x_mean = X.mean()
    x_std = X.std()

    X = (X - x_mean) / x_std
    X["intercept"] = 1
    X = X[["intercept"] + predictors]

    penalty = alpha * np.identity(X.shape[1])
    penalty [0][0] = 0

    B = np.linalg.inv(X.T @ X + penalty) @ X.T @ y
    B.index = ['intersept', 'duration_ms', 'danceability' , 'energy', 'key',
               'loudness', 'mode','speechiness', 'acousticness', 'instrumentalness', 'liveness',
               'valence', 'tempo', 'time_signature']

    return B, x_mean, x_std
```

```
def ridge_predict(test, predictors, x_mean, x_std, B):
    test_X = test[predictors]
    test_X = (test_X - x_mean) / x_std
    test_X["intercept"] = 1
    test_X = test_X[["intercept"] + predictors]

    predictions = np.dot( test_X, B)
    return predictions
```

```
errors = []
alphas = [1, 100, 10000, 1000000]
```

```
print(alphas)
```

```
[1, 100, 10000, 1000000]
```

```
for alpha in alphas:
    B, x_mean, x_std = ridge_fit(train, predictors, target, alpha)
    predictions = ridge_predict(test, predictors, x_mean, x_std, B)
    errors.append(mean_absolute_error(test[target], predictions))
```

```
print(errors) #we choose alpha = 1
```

```
[18.4342503776153, 18.43479886564248, 18.486193234055335, 18.884534436491602]
```

```
#cross validation for numeric model with alpha = 1
```

```
ridge = Ridge(alpha = 1)
```

```
ridge.fit(X[predictors], y)
```

```
Ridge(alpha=1)
```

```
ridge.coef_
```

```
array([[-0.23549972,  1.54551487, -0.76655982, -0.08949634,  0.62831129,
        -0.39275657, -1.30666316, -0.26108605, -2.52582112,  0.32743025,
        -2.48987922,  0.42300919,  0.464909  ]])
```

```
ridge.intercept_
```

```
array([33.23557018])
```

```
kf = KFold(n_splits = 5, random_state = 677, shuffle = True)
kf.get_n_splits(X)
for i, (train_index, test_index) in enumerate(kf.split(X)):
    print(f'Fold {i}')
    print(f' Train: index={train_index}')
    print(f' Test: index={test_index}')
```

```
Fold 0
 Train: index=[    0     1     2 ... 91196 91197 91198]
 Test: index=[    5     7     8 ... 91188 91195 91199]
Fold 1
 Train: index=[    0     2     3 ... 91196 91197 91199]
 Test: index=[    1     6    13 ... 91178 91194 91198]
Fold 2
 Train: index=[    0     1     2 ... 91197 91198 91199]
 Test: index=[    9    10    21 ... 91190 91191 91192]
Fold 3
 Train: index=[    0     1     2 ... 91195 91198 91199]
 Test: index=[    4    11    12 ... 91181 91196 91197]
Fold 4
 Train: index=[    1     4     5 ... 91197 91198 91199]
 Test: index=[    0     2     3 ... 91179 91184 91193]
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20)
y_test.shape
```

```
(18240, 1)
```

```
scores = cross_val_score(ridge, X, y, scoring = "neg_mean_squared_error", cv = kf, n_jobs = -1)
print(scores)
```

```
[-481.60937937 -488.95538665 -487.31206431 -484.0066355  -486.76297418]
```

```
np.sqrt(np.mean(np.absolute(scores)))
```

```
22.039266957021503
```

```
#ridge regression for every variable in dataset
```

```
# checking what are the categories in two variables: explicit and track_genre
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 114000 entries, 0 to 113999
Data columns (total 21 columns):
 #   Column            Non-Null Count   Dtype  
---  ------            --------------   -----  
 0   Unnamed: 0        114000 non-null  int64  
 1   track_id          114000 non-null  object 
 2   artists           113999 non-null  object 
 3   album_name        113999 non-null  object 
 4   track_name        113999 non-null  object 
 5   popularity        114000 non-null  int64  
 6   duration_ms       114000 non-null  int64  
 7   explicit          114000 non-null  bool   
 8   danceability      114000 non-null  float64
 9   energy            114000 non-null  float64
 10  key               114000 non-null  int64  
 11  loudness          114000 non-null  float64
 12  mode              114000 non-null  int64  
 13  speechiness       114000 non-null  float64
 14  acousticness      114000 non-null  float64
 15  instrumentalness  114000 non-null  float64
 16  liveness          114000 non-null  float64
 17  valence           114000 non-null  float64
 18  tempo             114000 non-null  float64
 19  time_signature    114000 non-null  int64  
 20  track_genre       114000 non-null  object 
dtypes: bool(1), float64(9), int64(6), object(5)
memory usage: 17.5+ MB
```

```
data.track_genre.unique()
```

```
array(['acoustic', 'afrobeat', 'alt-rock', 'alternative', 'ambient',
       'anime', 'black-metal', 'bluegrass', 'blues', 'brazil',
       'breakbeat', 'british', 'cantopop', 'chicago-house', 'children',
       'chill', 'classical', 'club', 'comedy', 'country', 'dance',
       'dancehall', 'death-metal', 'deep-house', 'detroit-techno',
       'disco', 'disney', 'drum-and-bass', 'dub', 'dubstep', 'edm',
       'electro', 'electronic', 'emo', 'folk', 'forro', 'french', 'funk',
       'garage', 'german', 'gospel', 'goth', 'grindcore', 'groove',
       'grunge', 'guitar', 'happy', 'hard-rock', 'hardcore', 'hardstyle',
       'heavy-metal', 'hip-hop', 'honky-tonk', 'house', 'idm', 'indian',
       'indie-pop', 'indie', 'industrial', 'iranian', 'j-dance', 'j-idol',
       'j-pop', 'j-rock', 'jazz', 'k-pop', 'kids', 'latin', 'latino',
       'malay', 'mandopop', 'metal', 'metalcore', 'minimal-techno', 'mpb',
       'new-age', 'opera', 'pagode', 'party', 'piano', 'pop-film', 'pop',
       'power-pop', 'progressive-house', 'psych-rock', 'punk-rock',
       'punk', 'r-n-b', 'reggae', 'reggaeton', 'rock-n-roll', 'rock',
       'rockabilly', 'romance', 'sad', 'salsa', 'samba', 'sertanejo',
       'show-tunes', 'singer-songwriter', 'ska', 'sleep', 'songwriter',
       'soul', 'spanish', 'study', 'swedish', 'synth-pop', 'tango',
       'techno', 'trance', 'trip-hop', 'turkish', 'world-music'],
      dtype=object)
```

```
data.track_genre.value_counts()
```

```
acoustic             1000
punk-rock            1000
progressive-house    1000
power-pop            1000
pop                  1000
                     ...
folk                 1000
emo                  1000
electronic           1000
electro              1000
world-music          1000
Name: track_genre, Length: 114, dtype: int64
```

```
data.explicit.unique()
data.explicit.value_counts()
```

```
False    104253
True       9747
Name: explicit, dtype: int64
```

```
# explicit is a boolean and track_genre is categorical data that has to be changed using one hot encoding
```

```
data['explicit'] = data['explicit'].astype('category')
data['track_genre'] = data['track_genre'].astype('category')
```

```
data['e_new'] = data['explicit'].cat.codes
data['tg_new'] = data['track_genre'].cat.codes
```

```
enc = OneHotEncoder()
```

```
enc_data = pd.DataFrame(enc.fit_transform(
    data[['e_new', 'tg_new']]).toarray())
```

```
new_df = data.join(enc_data)
```

```
print(new_df)
```

```
        Unnamed: 0                track_id                 artists  \
0                0  5SuOikwiRyPMVoIQDJUgSV              Gen Hoshino
1                1  4qPNDBW1i3p13qLCt0Ki3A             Ben Woodward
2                2  1iJBSr7s7jYXzM8EGcbK5b  Ingrid Michaelson;ZAYN
3                3  6lfxq3CG4xtTiEg7opyCyx             Kina Grannis
4                4  5vjLSffimiIP26QG5WcN2K         Chord Overstreet
...            ...                     ...                     ...
113995      113995  2C3TZjDRiAzdyViavDJ217            Rainy Lullaby
113996      113996  1hIz5L4IB9hN3WRYPOCGPw            Rainy Lullaby
113997      113997  6x8ZfSoqDjuNa5SVP5QjvX            Cesária Evora
113998      113998  2e6sXL2bYv4bSz6VTdnfLs         Michael W. Smith
113999      113999  2hETkH7cOfqmz3LqZDHZf5            Cesária Evora

                                           album_name  \
0                                              Comedy
1                                    Ghost (Acoustic)
2                                      To Begin Again
```

```
3       Crazy Rich Asians (Original Motion Picture Sou...
4                                                 Hold On
...                                                   ...
113995  #mindfulness - Soft Rain for Mindful Meditatio...
113996  #mindfulness - Soft Rain for Mindful Meditatio...
113997                                            Best Of
113998                                  Change Your World
113999                                     Miss Perfumado

                      track_name  popularity  duration_ms explicit  \
0                         Comedy          73       230666    False
1               Ghost - Acoustic          55       149610    False
2                 To Begin Again          57       210826    False
3       Can't Help Falling In Love          71       201933    False
4                        Hold On          82       198853    False
...                          ...         ...          ...      ...
113995         Sleep My Little Boy          21       384999    False
113996            Water Into Light          22       385000    False
113997              Miss Perfumado          22       271466    False
113998                     Friends          41       283893    False
113999                   Barbincor          22       241826    False

        danceability  energy  ...  106  107  108  109  110  111  112  113  \
0              0.676  0.4610  ...  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
1              0.420  0.1660  ...  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
2              0.438  0.3590  ...  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
3              0.266  0.0596  ...  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
4              0.618  0.4430  ...  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
...              ...     ...  ...  ...  ...  ...  ...  ...  ...  ...  ...
113995         0.172  0.2350  ...  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
113996         0.174  0.1170  ...  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
113997         0.629  0.3290  ...  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
113998         0.587  0.5060  ...  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
113999         0.526  0.4870  ...  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

        114  115
0       0.0  0.0
1       0.0  0.0
2       0.0  0.0
3       0.0  0.0
```

```
#Categorical variables were changed usinh one-hot encoding the model with all the variables is ready to be built


#Rdge regression for alpha = 1


train, test = train_test_split(new_df, test_size=.2, random_state = 1)


predictors_f = ['duration_ms', 'danceability' , 'energy', 'key',
              'loudness', 'mode','speechiness', 'acousticness', 'instrumentalness', 'liveness',
              'valence', 'tempo', 'time_signature', 'tg_new', 'e_new']
target_f = 'popularity'


X_f = train[predictors_f].copy()
y_f = train[[target_f]].copy()


x_mean_f = X_f.mean()
x_std_f = X_f.std()


X_f = (X_f - x_mean_f) / x_std_f


X_f.describe()
```

duration_ms danceability energy key loudness

```
X_f["intercept"] = 1
X_f = X_f[["intercept"] + predictors_f]
```

```
X_f.T
```

	104483	17411	73414	95288	77403	1216	6
intercept	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.00
duration_ms	0.066370	-0.382093	0.900388	0.450949	-0.123068	-0.417338	0.28
danceability	-0.867615	0.645814	1.134945	1.129191	-0.125286	0.116402	0.61
energy	-1.124514	1.233915	-1.454058	0.666145	0.169843	0.177783	0.89
key	1.316531	-1.210086	-1.490821	-1.490821	0.474325	1.035796	0.47
loudness	-0.053266	1.087277	-0.699006	0.655463	0.075882	0.624563	0.97
mode	0.756612	0.756612	-1.321667	0.756612	0.756612	-1.321667	0.75
speechiness	-0.364595	-0.237009	-0.429340	-0.368403	-0.371260	0.357120	0.82
acousticness	0.482657	-0.948229	-0.214774	-0.274897	1.005730	-0.935413	-0.87
instrumentalness	-0.504215	1.710293	1.855560	-0.504215	-0.504215	-0.481456	-0.50
liveness	-0.538002	1.582232	0.369920	-0.699119	2.626605	-0.895922	-0.61
valence	-1.380735	-1.153252	-0.143070	0.828556	1.256534	0.909525	-0.17
tempo	-1.381716	0.199269	-0.235327	-1.182783	1.268994	2.064323	0.19
time_signature	-2.101340	0.221990	0.221990	0.221990	0.221990	0.221990	0.22
tg_new	1.440349	-1.203087	0.498435	1.166890	0.619972	-1.689236	0.16
e_new	-0.306267	-0.306267	-0.306267	-0.306267	-0.306267	-0.306267	-0.30

16 rows × 91200 columns

```
alpha = 1
I_f = np.identity(X_f.shape[1])
```

```
print(I_f)
```

```
[[1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]]
```

```
I_f [0][0] = 0
```

```
penalty_f = alpha * I_f
```

```
B_f = np.linalg.inv(X_f.T @ X_f + penalty_f) @ X_f.T @ y_f
```

```
#Adding raw labels into a df
B_f.index = ['intersept', 'duration_ms', 'danceability' , 'energy', 'key',
              'loudness', 'mode','speechiness', 'acousticness', 'instrumentalness', 'liveness',
              'valence', 'tempo', 'time_signature', 'track_genre', 'explicit']
print(B_f)
```

```
                  popularity
intersept          33.235570
duration_ms        -0.176065
danceability        1.448739
energy             -0.738013
key                -0.080412
loudness            0.606937
```

```
mode               -0.373040
speechiness        -1.532344
acousticness       -0.218127
instrumentalness   -2.411360
liveness            0.313431
valence            -2.415817
tempo               0.445081
time_signature      0.465459
track_genre         0.602375
explicit            0.976242
```

```
test_X_f = test[predictors_f]
test_X_f = (test_X_f - x_mean_f)/x_std_f
test_X_f["intercept"] = 1
test_X_f = test_X_f[["intercept"] + predictors_f]
print(test_X_f)
```

```
       intercept  duration_ms  danceability    energy       key  loudness  \
21719          1     0.110035      1.629831  0.074552  1.597266  0.677450
25741          1    -0.176540      0.421389  0.463653  1.035796  0.189382
15850          1    -0.603221      0.835712 -1.168188 -1.490821 -0.415356
31992          1     0.071208      0.686096  1.075098  1.316531  0.961695
79027          1    -0.287700     -1.753805  0.142050  0.474325  0.519184
...          ...          ...           ...       ...       ...       ...
58712          1     0.496233      0.087629  0.646293 -1.490821  0.665764
67442          1     0.204075     -0.620172 -0.417779 -0.087145  0.496405
36107          1    -0.501106      1.169472  0.928192  1.316531  0.842649
40989          1     0.415257     -0.562627  0.360423 -0.367880  0.709143
30180          1    -0.422800      0.030085  0.431890 -1.210086  0.638627

           mode  speechiness  acousticness  instrumentalness  liveness  \
21719 -1.321667     0.804622      0.305293         -0.504049 -0.023688
25741 -1.321667    -0.161791      0.545787         -0.504215 -0.854987
15850 -1.321667    -0.414106      0.188053         -0.435778 -0.538002
31992  0.756612    -0.152270     -0.788050         -0.504215  0.936715
79027  0.756612    -0.301754      0.395478         -0.504215 -0.044680
...         ...          ...           ...               ...       ...
58712  0.756612    -0.297946     -0.945318         -0.152347 -0.618298
67442  0.756612    -0.515031      0.073818         -0.504215 -0.023688
36107 -1.321667    -0.026588      0.179034         -0.504215  0.637573
40989  0.756612    -0.459808      0.121917         -0.504215 -0.690197
30180 -1.321667    -0.229392     -0.663896         -0.504215  0.863242

        valence     tempo  time_signature    tg_new     e_new
21719  0.570227 -0.670990         0.22199 -1.081549  3.265087
25741  1.445461 -0.503150         0.22199 -0.960012 -0.306267
15850  0.099837  0.464223         0.22199 -1.263855 -0.306267
31992  0.142249 -0.737546         0.22199 -0.777706 -0.306267
79027 -0.420677  2.849644        -2.10134  0.680741 -0.306267
...         ...       ...             ...       ...       ...
58712 -0.802387 -0.061215        -2.10134  0.042670 -0.306267
67442 -1.045293 -1.536645         0.22199  0.316129 -0.306267
36107  1.125442 -0.269222         0.22199 -0.625785 -0.306267
40989 -0.127647  0.594531         0.22199 -0.504247 -0.306267
30180  0.003445  0.734648         0.22199 -0.808091  3.265087

[22800 rows x 16 columns]
```

```
predictions_f = np.dot( test_X_f , B_f)
print(predictions_f)
```

```
[[37.16974194]
 [30.64650954]
 [36.25963805]
 ...
 [33.23770433]
 [34.01018379]
 [39.11825349]]
```

```
print(predictions_f.mean())
print(predictions_f.std())
print(predictions_f.min())
print(predictions_f.max())
```

```
33.2217497445891
3.5681891667597325
12.49818037857034
45.49907629893511
```

```
# Ridge Regression for alpha = 100
```

```
alpha2= 100
penalty2_f= alpha2 * I_f
```

```
B2_f = np.linalg.inv(X_f.T @ X_f + penalty2_f) @ X_f.T @ y_f
B2_f.index = ['intersept', 'duration_ms', 'danceability' , 'energy', 'key',
              'loudness', 'mode','speechiness', 'acousticness', 'instrumentalness', 'liveness',
              'valence', 'tempo', 'time_signature', 'track_genre', 'explicit']
predictions2_f = np.dot( test_X_f , B2_f)
print(B2_f)
print(predictions2_f)
```

```
                  popularity
intersept          33.235570
duration_ms        -0.175651
danceability        1.444691
energy             -0.737460
key                -0.080386
loudness            0.607369
mode               -0.372680
speechiness        -1.529593
acousticness       -0.218082
instrumentalness   -2.407490
liveness            0.312105
valence            -2.410154
tempo               0.443861
time_signature      0.465241
track_genre         0.601954
explicit            0.975218
[[37.16451231]
 [30.65285102]
 [36.25335707]
 ...
 [33.23756783]
 [34.01013512]
 [39.1102812 ]]
```

```
print(predictions2_f.mean())
print(predictions2_f.std())
print(predictions2_f.min())
print(predictions2_f.max())
```

```
33.22175653031229
3.5623946451075996
12.529684019329943
45.472330879871066
```

```
# Ridge Regression for alpha = 10 000


alpha3= 10000
penalty3_f= alpha3 * I_f


B3_f = np.linalg.inv(X_f.T @ X_f + penalty3_f) @ X_f.T @ y_f
B3_f.index = ['intersept', 'duration_ms', 'danceability' , 'energy', 'key',
              'loudness', 'mode','speechiness', 'acousticness', 'instrumentalness', 'liveness',
              'valence', 'tempo', 'time_signature', 'track_genre', 'explicit']
predictions3_f = np.dot( test_X_f , B3_f)
print(B3_f)
print(predictions3_f)
```

```
                  popularity
intersept          33.235570
duration_ms        -0.144099
danceability        1.134023
energy             -0.648026
key                -0.076781
loudness            0.605104
mode               -0.337745
speechiness        -1.302629
acousticness       -0.201247
instrumentalness   -2.088528
liveness            0.207124
valence            -1.957090
tempo               0.347901
time_signature      0.439734
track_genre         0.560556
explicit            0.880643
[[36.7088942 ]
 [31.18258046]
 [35.72258495]
 ...
 [33.25227765]
 [33.99237875]
 [38.4096096 ]]
```

```
print(predictions3_f.mean())
print(predictions3_f.std())
```

```
print(predictions3_f.min())
print(predictions3_f.max())
```

```
33.22245441059542
3.0784637016929968
15.228763694813729
43.44904011344275
```

```
# Ridge Regression for alpha = 1 000 000
```

```
alpha4= 1000000
penalty4_f= alpha4 * I_f
```

```
B4_f = np.linalg.inv(X_f.T @ X_f + penalty4_f) @ X_f.T @ y_f
B4_f.index = ['intersept', 'duration_ms', 'danceability' , 'energy', 'key',
              'loudness', 'mode','speechiness', 'acousticness', 'instrumentalness', 'liveness',
              'valence', 'tempo', 'time_signature', 'track_genre', 'explicit']
predictions4_f = np.dot( test_X_f , B4_f)
print(B4_f)
print(predictions4_f)
```

```
                  popularity
intersept          33.235570
duration_ms        -0.014442
danceability        0.068935
energy             -0.009696
key                -0.006735
loudness            0.086198
mode               -0.029269
speechiness        -0.084175
acousticness       -0.035813
instrumentalness   -0.176372
liveness           -0.007016
valence            -0.081515
tempo               0.020876
time_signature      0.054998
track_genre         0.060695
explicit            0.080193
[[33.59024664]
 [33.20075523]
 [33.34738859]
 ...
 [33.34946448]
 [33.33672257]
 [33.70745614]]
```

```
print(predictions4_f.mean())
print(predictions4_f.std())
print(predictions4_f.min())
print(predictions4_f.max())
```

```
33.23465191721801
0.2867985959689753
31.672264979291654
33.98837548506509
```

```
#errors for every model built
```

```
def ridge_fit_f(train, predictors_f, target_f, alpha_f):
    X_f = train[predictors_f].copy()
    y_f = train[[target_f]].copy()

    x_f_mean = X_f.mean()
    x_f_std = X_f.std()

    X_f = (X_f - x_f_mean) / x_f_std
    X_f["intercept"] = 1
    X_f = X_f[["intercept"] + predictors_f]

    penalty_f = alpha_f * np.identity(X_f.shape[1])
    penalty_f [0][0] = 0

    B_f = np.linalg.inv(X_f.T @ X_f + penalty_f) @ X_f.T @ y_f
    B_f.index = ['intersept', 'duration_ms', 'danceability' , 'energy', 'key',
              'loudness', 'mode','speechiness', 'acousticness', 'instrumentalness', 'liveness',
              'valence', 'tempo', 'time_signature', 'track_genre', 'explicit']

    return B_f, x_f_mean, x_f_std
```

```
def ridge_predict_f(test, predictors_f, x_mean_f, x_std_f, B_f):
    test_X_f = test[predictors_f]
    test_X_f = (test_X_f - x_f_mean) / x_f_std
    test_X_f["intercept"] = 1
    test_X_f = test_X_f[["intercept"] + predictors_f]

    predictions_f = np.dot( test_X_f, B_f)
    return predictions_f


errors_f = []
alphas_f = [1, 100, 10000, 1000000]


for alpha_f in alphas_f:
    B_f, x_f_mean, x_f_std = ridge_fit_f(train, predictors_f, target_f, alpha_f)
    predictions_f = ridge_predict_f(test, predictors_f, x_mean_f, x_std_f, B_f)
    errors_f.append(mean_absolute_error(test[target_f], predictions_f))


print(errors_f)
```

```
[18.389921118338037, 18.39045698350531, 18.441395258129226, 18.879089369915096]
```

```
ridge_f = Ridge(alpha =1)
```

```
ridge_f.fit(X_f[predictors_f], y_f)
```

```
Ridge(alpha=1)
```

```
ridge_f.coef_
```

```
array([[-0.17606547,  1.44873869, -0.7380133 , -0.08041214,  0.60693748,
        -0.3730399 , -1.53234426, -0.21812679, -2.41135953,  0.31343052,
        -2.4158174 ,  0.44508072,  0.46545919,  0.60237505,  0.97624153]])
```

```
scores_f = cross_val_score( ridge_f, X_f, y_f, scoring = "neg_mean_squared_error", n_jobs = -1)
print(scores_f)
```

```
[-486.84832371 -482.25616423 -486.65312364 -485.66652677 -481.61508803]
```

```
np.sqrt(np.mean(np.mean(np.absolute(scores_f))))
```

```
22.013810330708605
```